BLOCK MAKER LIVE

Guía de ejercicios

Módulo 8

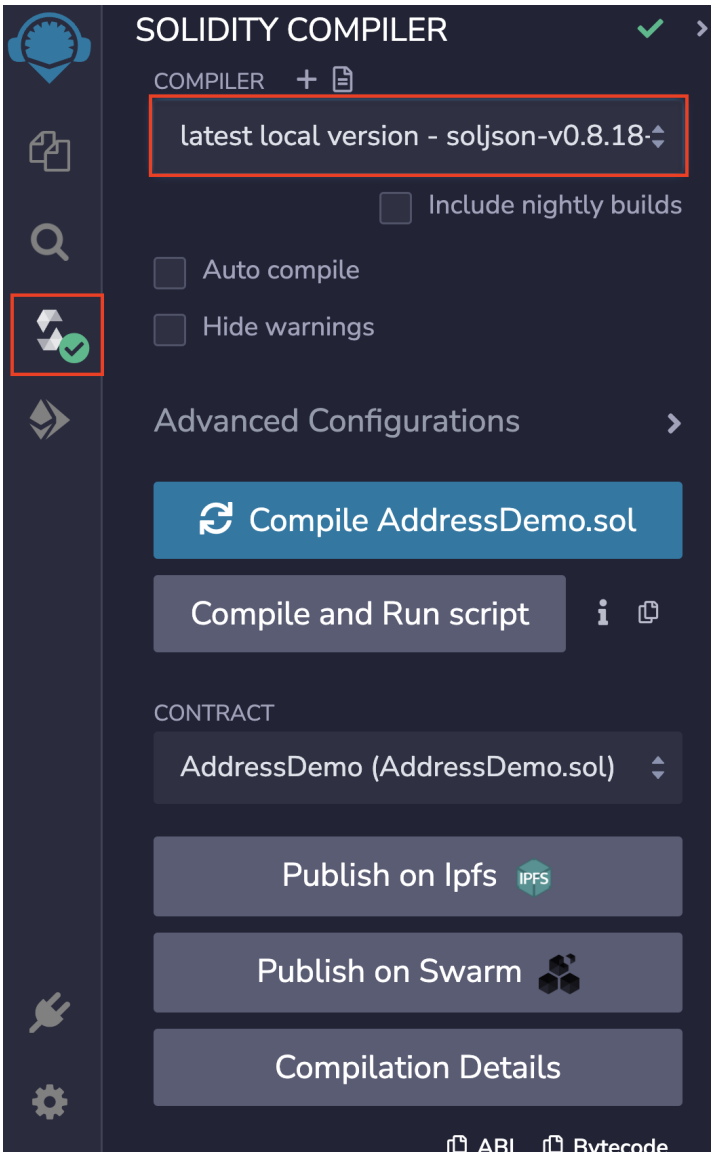
Recordatorios

1.A continuación, os dejamos un enlace a modo de bibliografía y de utilidad para consultar cualquier duda en materia de Solidity (si lo necesitáis, usar la función de traducir página al español):

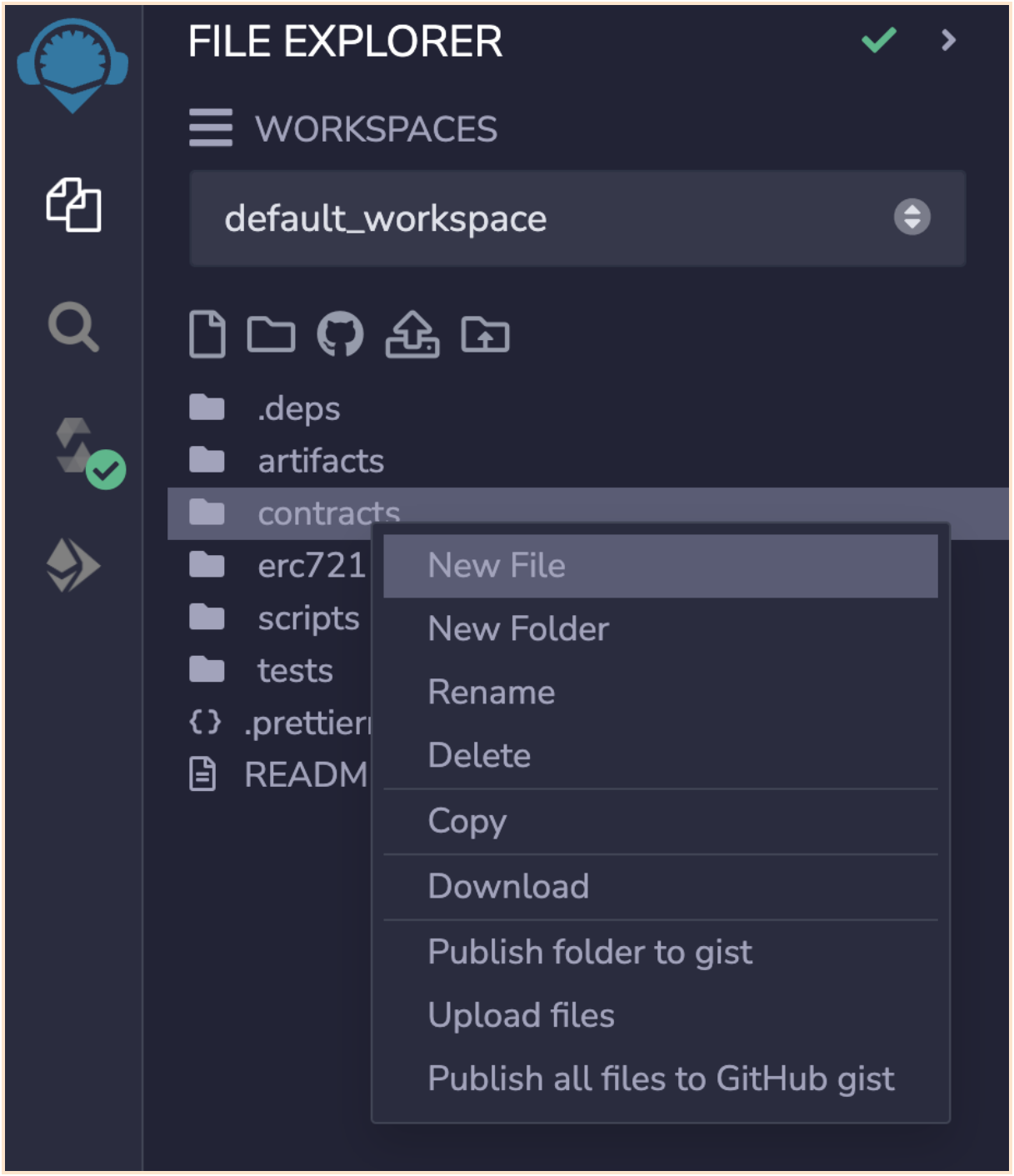
- https://docs.soliditylang.org/en/v0.8.21/grammar.html

2.Para realizar estos ejercicios vamos a utilizar el compilador online de Remix http://remix.ethereum.org/

RECUERDA 1: Desde la página http://remix.ethereum.org/ vamos a irnos a la tercera sección de la izquierda y vamos a cambiar la versión del compilador. Para ello, seleccionaremos cualquiera superior a la 0.8.0:



RECUERDA 2: Para crear un nuevo contrato, nos iremos a la primera sección de la izquierda y colocaremos en la carpeta “contracts” -> “New File” y le pondremos una nombre con la extensión “.sol.



Paso 0. Crear el contrato

Crea un nuevo contrato Solidity con el nombre "**tokenERC20**"

OBJETIVO: Implementación de un Contrato ERC-20

Paso 1. Base del ejercicio

Dada la siguiente base del ejercicio, se requiere que se rellenen las partes del código que falta en las funciones (Donde se indica rellenar). Se puede copiar y pegar en REMIX:

```
// SPDX-License-Identifier: MIT

// Versión del compilador de Solidity.
pragma solidity ^0.8.0;

// Importa la interfaz ERC20 del paquete OpenZeppelin.
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

// Definición del contrato MyToken que implementa la interfaz ERC20.
contract MyToken is IERC20 {
      // Variables para almacenar el nombre, símbolo y decimales del
token.
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    // Mappings para almacenar balances y asignaciones permitidas.
    mapping(address => uint256) private _balances;
          mapping(address   =>   mapping(address   =>   uint256))   private
_allowances;

    // Variable para almacenar el suministro total de tokens.
    uint256 private _totalSupply;

      // Constructor del contrato que establece el nombre, símbolo y
suministro total.
    constructor(string memory name_, string memory symbol_) {
```

```
        _name = name_;
        _symbol = symbol_;
        _decimals = 18;
        _totalSupply = 1000000 * 10**uint256(_decimals);
        _balances[msg.sender] = _totalSupply;
        emit Transfer(address(0), msg.sender, _totalSupply);
    }

        // Funciones para obtener el nombre, símbolo, decimales y
suministro total del token.
    function name() public view returns (string memory) {
        //Rellenar
    }

    function symbol() public view returns (string memory) {
        //Rellenar
    }

    function decimals() public view returns (uint8) {
        //Rellenar
    }

    function totalSupply() public view override returns (uint256) {
        //Rellenar
    }

     // Funciones para obtener el balance, realizar transferencias, y
consultar asignaciones permitidas.
     function balanceOf(address account) public view override returns
(uint256) {
        //Rellenar
    }

        function transfer(address recipient, uint256 amount) public
override returns (bool) {
        //Rellenar
    }

      function allowance(address owner, address spender) public view
override returns (uint256) {
        //Rellenar
    }
```

```
    // Funciones para aprobar transferencias y realizar transferencias desde.
    function approve(address spender, uint256 amount) public override returns (bool) {
        //Rellenar
    }

    function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {
        //Rellenar
    }

    // Funciones para aumentar y disminuir asignaciones permitidas.
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        //Rellenar
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
        //Rellenar
    }

    // Función interna para realizar la transferencia de tokens.
    function _transfer(address sender, address recipient, uint256 amount) internal {
        //Rellenar
    }

    // Función interna para aprobar asignaciones permitidas.
    function _approve(address owner, address spender, uint256 amount) internal {
        //Rellenar
    }
}
```

Paso 2. Pistas

Os podéis ayudar con la información por ejemplo proporcionada por OpenZepellin en https://github.com/OpenZeppelin/openzeppelin-contracts/tree/master/contracts/token/ERC20
o
https://docs.openzeppelin.com/contracts/5.x/api/token/erc20
Más las propias búsquedas que realiceis de información.

A continuación se indica lo que ha de hacer cada una de las funciones:

```
function name()
```

Devolver el nombre del token.

```
function symbol()
```

Devolver el símbolo del token.

```
function decimals()
```

Devolver la cantidad de decimales del token.

```
function totalSupply()
```

Devolver el suministro total del token.

```
function balanceOf()
```

Devolver el balance de la dirección proporcionada.

```
function transfer()
```

Llamar a la función de transferencia interna. `_transfer`

```
function allowance()
```

Devolver la asignación permitida para el spender desde el owner.

```
function approve()
```

Llamar a la función interna de aprobación.`_approve`

```
function transferFrom()
```

Llamar a la función de transferencia interna.`_transfer`
Actualizar la asignación permitida.`_approve`

```
function increaseAllowance()
```

Aumentar la asignación permitida..`_approve`

```
function decreaseAllowance()
```

Disminuir la asignación permitida..`_approve`

```
function _transfer()
```

Verificar que el remitente no sea la dirección cero.
Verificar que el destinatario no sea la dirección cero.
Restar el monto del remitente.`_balances`
Sumar el monto al destinatario.`_balances`
Emitir el evento de transferencia.`emit Transfer`

```
function _approve()
```

Verificar que el propietario no sea la dirección cero.
Verificar que el spender no sea la dirección cero.
Establecer la asignación permitida.`_allowances`
Emitir el evento de aprobación.`emit Approval`